Computação Paralela e Distribuída
Trabalho Prático nº1

Group: t04g07

Alberto José Ribeiro da Cunha : up201906325
Gustavo Speranzini Tosi Tavares : up201700129

FEUP : LEIC
28 de Março de 2022

Introdução

O primeiro trabalho de CPD consistia em comparar o impacto que o acesso a uma grande quantidade de dados tem no desempenho da hierarquia de memória do processador. Para tal recorremos à multiplicação de duas matrizes, usando diferentes algoritmos em diferentes linguagens de programação. As linguagens escolhidas foram C/C++ e Java. Utilizamos algoritmos de multiplicação de matrizes normais, em bloco e por linha. As métricas de performance utilizados foram o tempo que demora o algoritmo a chegar ao resultado, pois é um parâmetro bastante óbvio para avaliar qual o algoritmo mais eficiente, gflops, o número de operações flutuantes por segundo, dá uma ideia de como o algoritmo está a conseguir aceder a memória. Para além destes, para o C/C++, têm-se os *data cache misses*, DCM, dos algoritmos. Isto permite ver o quanto é que os algoritmos têm que aceder a dados que não estão em cache, o que aumenta o tempo que o processo demora.

Algoritmos

1. Normal

Este é o algoritmo comum do cálculo do produto de matrizes. Percorre-se as matrizes, multiplicando os elementos de uma linha da matriz A pelos elementos de uma coluna da matriz B. Faz-se o somatório destes produtos e obtém-se um elemento da matriz resultante.

Tamanho	Tempo	Gflops	L1 DCM	L2 DCM
600	0.233	1.852	244765813	39623437
1000	1.740	1.149	1227521884	169248552
1400	5.314	1.034	3541673898	1528827503
1800	22.142	0.527	9067046166	6236429590
2200	47.080	0.452	17641206871	24951644351
2600	81.949	0.429	30870916777	52753193450
3000	136.503	0.396	50289811940	97455369191

Resultados em C/C++

Tamanho	Tempo	Gflops
600	0.416	1.038
1000	3.046	0.657
1400	11.597	0.473
1800	33.734	0.346
2200	71	0.3
2600	122.217	0.288
3000	198.185	0.272

Resultado em Java

É possível de ver, segundo estes resultados, que, obviamente, quanto maior as matrizes forem mais tempo demoram. Isto também faz diminuir o número de flops. O número de cache misses de dados nos níveis 1 e 2 também aumentam com o tamanho. Isto tudo pode ser explicado pelo aumento de tamanho que gera um aumento no uso de memória. O aumento do tamanho leva a mais operações o que levará a demorar mais tempo e com mais memória necessária a probabilidade de necessitar de um elemento que não está em cache é maior. Como era de esperar os tempos em C/C++ são mais rápidos do que em Java. Isto acontece, pois, os compiladores em C/C++ tendem a só compilar o código sem quaisquer mecanismos de segurança como acontece em certas línguas como é o caso do Java.

2. **Linha**

Neste algoritmo é similar ao anterior, mas em vez de calcular um elemento de cada vez, multiplica-se já uma linha toda da matriz B por um elemento de A. Faz-se isso por toda a matriz de A e adiciona-se os valores no fim. Isto garante que os elementos de A são apenas acedidos uma vez.

Tamanho	Tempo	Gflops	L1 DCM	L2 DCM
4096	55.175	2.491	17596639879	17815206021
6144	186.376	2.489	59407557315	61184559518
8192	442.228	2.486	140726946475	152444026474
10240	870.252	2.468	274759129278	312476905050

Resultados em C/C++

Tamanho	Tempo	Gflops
600	0.25	1.728
1000	1.23	1.626
1400	4.625	1.187
1800	9.958	1.171
2200	17.875	1.191
2600	29.526	1.191
3000	49.298	1.095

Resultado em Java

Comparando os resultados deste algoritmo com o anterior, é evidente o aumento da velocidade de processamento e de número de operações de vírgula flutuante. Isto se deve ao facto de os elementos de A precisarem de ficar apenas uma vez na cache. São chamados pela 1ª vez, ficam na cache e depois é chamado o próximo, seguindo assim até ao fim. Isto leva a muitos menos erros, o que acelera o processo. Também é interessante que os Gflops continuam praticamente constantes em todos os testes.

3. Bloco

O último algoritmo testado divide as matrizes em blocos de tamanho iguais. Estes blocos funcionam como matrizes mais pequenas que fariam o produto entre si. Os resultados destes produtos são depois somados, em matriz, para obter a matriz final. Isto faz com que apenas sejam acedidas partes das matrizes de cada vez. Utilizamos a mesma estratégia para cálculo do produto dos blocos que no último algoritmo.

Tamanho	Tamanho Bloco	Tempo	Gflops	L1	L2
4096	128	45.023	3.053	9933585501	32303749985
4096	256	46.609	2.949	9158635367	22155182017
4096	512	46.751	2.940	8769316612	18787657653
6144	128	147.826	3.138	33501359758	108228005468
6144	256	135.036	3.435	30888406111	76982237273
6144	512	140.568	3.300	29662976560	63129719407
8192	128	427.049	2.573	79498862866	256130395123
8192	256	430.169	2.556	73357598745	164320368886
8192	512	385.797	2.850	70249378603	148812793510
10240	128	684.993	3.135	155098217754	492169864374
10240	256	630.238	3.407	143026295254	349751556117
10240	512	641.719	3.346	137147685260	306193857107

Resultados em C/C++

Neste algoritmo, para além do aprimoramento usado no anterior, este também conta com uma estratégia em bloco, que diminui o espaço onde a memória é acedida de cada vez. Isto levará a menos missões no acesso a cache o que tornará este um algoritmo mais rápido e eficiente que os anteriores. Também é visível que quanto maior o tamanho dos blocos, menos falhas na cache há. Isto conclui-se, que, apesar de os blocos serem maiores, como existem menos deles, a probabilidade de erros é menor.

Conclusão

Como foi possível observar pelos algoritmos anteriores, um melhor acesso a memória e a cache permite a um algoritmo acelerar o seu tempo de execução. Para diminuir o espaço onde se vai procurar por um elemento é utilizado o mesmo é possível melhorar o desempenho, levando a que haja menos falhas na cache o que aumenta o desempenho de um programa. Também existe o fato de que diferentes linguagens de programação demora mais ou menos que outras, por isso a escolha de linguagem também é um fator a decidir no melhoramento da eficiência de um programa.